Rendering Large (Volume) Datasets: A new Parallel Visualization System

Sascha Schneider
Fraunhofer IGD
Fraunhoferstr. 5
64283 Darmstadt, Germany
sschneid@igd.fhg.de

Thorsten May
Fraunhofer IGD
Fraunhoferstr. 5
64283 Darmstadt, Germany
tmay@igd.fhg.de

Michael Schmidt
Fraunhofer IGD
Fraunhoferstr. 5
64283 Darmstadt, Germany
mschmidt@igd.fhg.de

ABSTRACT

In this paper we describe a basis for a system that is able to compute actual scientific and realistic visualization methods in parallel. It is capable to integrate easily in modern VR renderers like for example *Open Inventor* [SGIa], *Coin* [Coi] and *OpenSG* [Vos02]. Our approach is advantageous for processing large datasets which usually are the result of physically based simulation algorithms and programs. Using our techniques it is even more feasible to manage similar visualization problems for other large amounts of data (e.g. medicinal CT-scans or large geometries) in the context of displaying interactively.

Keywords

Interactive, Parallel Visualization, Computational Fluid Dynamics, Large Volume Data, Visualization System

1. INTRODUCTION

Besides of the simulation of physical phenomena their performant and professional visualization comes more and more into the focus of modern scientific applications. Nowadays there are several powerful physical based simulation programs available on the software market (e.g. *Fluent* [Flu], *FemLab* [Fem], *Flovent* [Flo], *CFX* [CFX], etc.) which allow the user and developer to simulate and investigate nearly every kind of physical problem in high quality and detail. In the realisation concepts of these products mostly parallel approaches play an important role in the program architecture to gain major increases in perfomance.

Empirically these simulation programs are producing large amounts of result data which are often displayed only roughly or inperformant using simple visualization tools. These tools are mostly already integrated within the simulation programs themselves. There are only few visualization programs available which are completely independent of the underlying simulation system and/or data format, grid and glyph types (e.g. *VTK* [Mar96]). These independent tools offer a good general approach for the visualization problem. On the other hand they often lack in performance to process the large amounts of simulation data in reasonable timeframes.

Very large data sets ($> 256^3$ grid points) mainly cause problems due to the

- data is too large to load into main storage completely
- loading data in the hierarchically ordered memory (hard disk, cache, main memory) takes too much time for qualitative, quantitative and interactive rendering.
- costly calculations for some visualization methods.

For these reasons, techniques from the areas of

- data compression (e.g. wavelet methods)
- parallelization of program code (e.g. multi threading, OpenMP [OMP], MPI [MPI])
- hardware accelerated algorithms (e.g. 3D texturing)
- efficient algorithms / software design (e.g. object oriented programming)
- utilization of efficient software development tools (e.g. C/C++) and libraries (e.g. OpenSG [Vos02], OpenGL [SGIb], Qt [Tro])
- development of portable, functional, easy usable and extendible software

were developed and steadily enhanced up to this day.

2. RELATED WORK

There are many works ([Moo96], [Bar99], [Fre99], [Rus00]) in computer graphics that use methods from some of the above areas. But to our knowledge, there is no comparable and published work that covers all of them. On the contrary, our approach to this topic applies most advanced methods from all of these areas in order to create a visualization tool for very large scientific data that features high rendering quality in real time, a very good functionality and an easy handling.

On the area of displaying simulation results many visualization methods have established today, like for example iso-surfaces, stream- and time-lines and volume rendering. But compared to the massive parallel simulation program, the corresponding available visualization software is still noticable less performant. The reason for that is that it mostly works either sequentially or it is implemented as a post process which operates on the basis of an offline rendering concept.

The transfer of the principle of parallel programming from the simulation of physical problems to their interactive visualization lets the end user profit of significant accelerations. This approach enables the applications to process even larger amounts of data interactively. Interactive Visualization of scientific data is a classical area of computer graphics that is steadily and rapidly developing due to the increasing requirements on data volume, display quality, program functionality and usability.

Approaches to interactive visualization in the past partly based on completely (data pre-processing and rendering) parallelization of existing visualization algorithms. By utilizing modern graphics hardware (NVIDIA GeForce [NVI], ATI Radeon [ATI]) it's possible to shift the rendering process to them. The advantage is a faster rendering on standard PCs, that do not provide many processors for parallel execution.

3. ARCHITECTURE

The main target of the following paper is to present a general concept for an interactive parallel visualization of scientific simulation data making use of the sophisticated capabilities of modern graphic cards. In this concept a generalized description of visualization methods is defined. Based on this description an extension of the developed system with further visualization methods is easily realizable every time. For that reason the system can be supplemented rapidly with new visualization methods as soon as they appear.

Furthermore our approach is realized platform independently, to satisfy the need for software portability because of different computer system architectures which are available and in use nowadays. The used in-

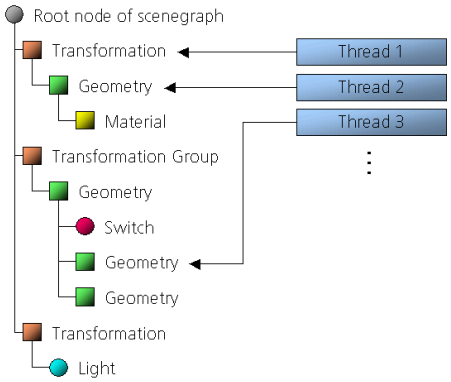

Figure 1: Example scenegraph with multiple threads modifing it.

gredients (C++, OpenGL, OpenSG and QT) allow us to fullfill this capability.

A rather new method in the field of scientific CFD visualization is to render scientific CFD data (stream lines, iso surfaces, ...) combined with detailed and textured geometrical data. This could be, for instance, a 3D model of the original scene that is used from the simulation side for generating the simulation mesh. Blending in original 3D models simplifies navigation for the user and is basis for another technical innovation, the application of highly realistic visualization methods within the area of CFD visualization. This means that in addition to visualization of abstract physical quantities like temperature, pressure, etc. with scientific visualization algorithms (cutting plane, glyphs, iso surface, ...), realistic quantities like fluid, gas or fire and smoke can be rendered as they appear in their natural form inside a photo realistic rendered virtual (simulation) environment [Sch02].

4. SCENEGRAPHS FUNDAMENTALS

Scenegraphs APIs (e.g. Open Inventor / Coin, OpenSG) are immediate APIs in which the objects and commands that are going to be rendered are not sent directly to the graphics processor (GPU) but are integrated in a (acyclic directed) tree graph based description of the displayed scene. This tree is then traversed and rendered separate from the application that provides the geometry and what shall be rendered. Therefore the scenegraph implements a kind of abstraction layer between the application and the GPU. Typical Objects in a scenegraph are normally derived from a base object ("'scene graph tree node"'): Geometry node, material node, transformation node, group node, light node.

Modern scenegraphs like OpenSG support parallel processing (fig. 1) natively so that it possible for the

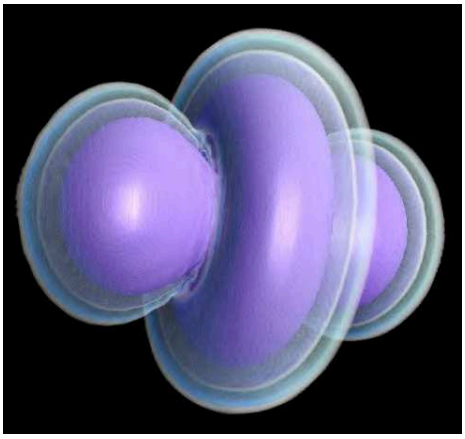

Figure 2: Visualization method: Iso surfaces.

application to modify parts or nodes of the scenegraph directly from several threads at the same time. As long as every application thread addresses a different node in the graph it is not necessary to lock the whole graph each time a part of it is accessed. In spite of this it is possible that every application thread can work exclusively and in parallel on its part of the scenegraph with no risk of creating an access conflict with other nodes / threads. These scenegraphs are called "'thread safe"'.

In this scenegraph we store the geometry of the surrounding (physical) scene (e.g. a car, an airplane or a tunnel) together with the data necessary for rendering the visualization methods which is after all render geometry (triangles, textures, colors, etc.) as well.

5. RENDERING

As mentioned in the beginning nowadays many methods for scientic visualization have established (e.g. iso-surfaces, stream lines, time lines, etc. - (see fig. 2, 3 and 4))

As every 3D rendering, these methods can be created in two ways: through ray tracing/casting on the one hand and through direct rendering using the capabilities of modern graphic cards (e.g. vertex and pixel shaders, shadowing, ...). To provide a parallel approach for the second method it is advisable to make use of thread safe scenegraphs in the first place. By doing so the application can process the calculation for each visualization method in parallel. After the calculation has finished each thread can store its calculated render information in a separate node in the scenegraph. Therefore it is easy to have several visualization methods at the same time in one displayed scene and to process each method in parallel. Furthermore each visualization method itself can be computed in parallel. By doing so, it only has to be assured, that the threaded calculations of one parallelized method itself

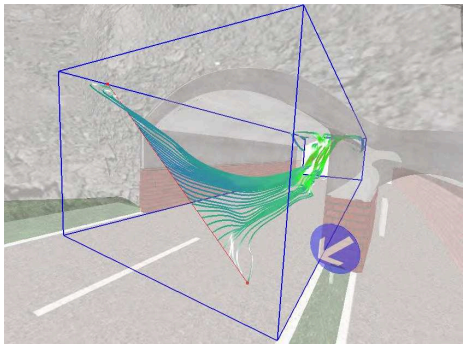

Figure 3: Visualization method: Stream lines.

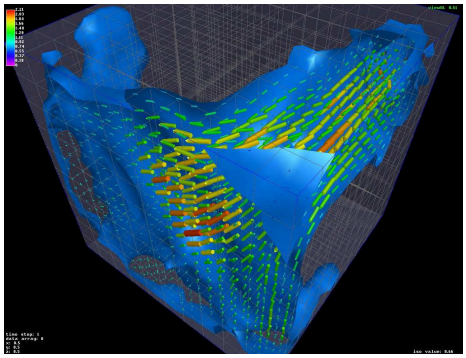

Figure 4: Visualization methods combined: iso-surface and vectors.

is brought together at the same time into the scenegraph to avoid flickering effects.

Probes

To allow the user to restrict visualization methods to certain areas of the datafield/scene we implemented the probe concept [Bar99]. The user can create as many of these probes (fig. 5) and place them in the scene as he wants. Each probe is associated with a cube in VR. In this cube only one certain visualization method is calculated. These probes can be placed arbitrarily and resized within the scene so that it is possible to let the system render the desired visualization method at every place in the dataset where the user wants it to be. To have two or more visualization methods displayed at the same location at the same time it is only necessary to place the corresponding probes together at the same coordinates in VR space.

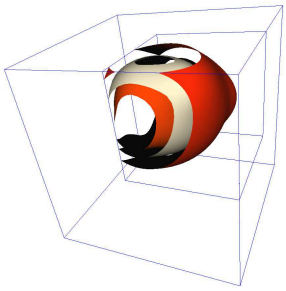

Figure 5: Iso-surfave probe (small cube) placed in the scene (big cube).

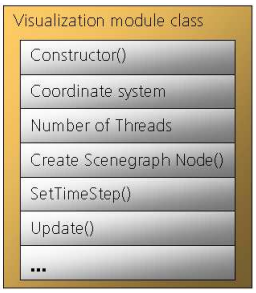

Figure 6: Basic functions and attributes of the general visualization module class.

Class Hierarchy

To allow an easy extension of the system afterwards we implemented a basic class of a visualization method. Based on this generalized description we implemented all visualization methods we needed. If the user wants to introduce a new (specialised) visualization method to the system, the only thing he has to do, is to inherit from the base (= *"'root"'*) class and implement its basic functions (calculation, node generation function, ...). Afterwards he only has to make sure to register his new visualization method within the system. Then he can start right off using his new method.

As shown in fig. 6 the basic functions of the general visualization module class consist of a class constructor, a coordinate system, the number of threads which will be used for calculations, a function to create a scenegraph node for the corresponding VR System, a function for setting the timestep which is used and a function to update the node = (re-)start the calculations. Now every new visualization method which is introduced to the visualization system has to inherit from this class and implement the derived functions according to their functionality.

Additionally each visualization method has to provide a user interface ("'panel-window"') (derived again from a basic generalized class) to the system together with a list of corresponding actions / commands. Every time a feature of an activated visualization method is changed (e.g. the user moves a slider in the interface for the color distribution of a certain method) an action transporting the changed parameters is sent through a inter-thread communication framework (by generating events which are collected in queues) to the central scheduler (see following section). This scheduler now interprets the upcoming events and assigns them to the corresponding active visualization modules. The interpretration of the incoming actions and events is implemented in the visualization module classes and not in the scheduler. This encapsulation allows the scheduler to be as general and flexible as possible. Furthermore it is of course possible to generate these kind of parameter changes not just by hand but by automated control over time for example.

6. PARALLEL CONCEPT

In this section the basic system layout (fig. 7) of the parallel visualization system is introduced showing how the different central parts of the software work together. The system is designed so that it is capable to check the features of the hardware platform it is running on (e.g. count the number of available CPUs or the amount of memory). Each visualization method is implemented fully scalable so that is possible to adjust the number of used calculation threads automatically by the system.

System Design

As one can see in fig. 7 user interacts with the scene and changes the parameters of the visualization. He has influence on the view of the scene (i.e. which part of the scene is rendered from which perspective). Additionally he can create and modify probes each carrying one certain visualization method. On the other side we have the kernel - we call it *"'central scheduler"'*, which collects all incoming actions / events and processes them (see fig. 8).

Inheritance

As mentioned in section each visualization method is derived from a basic class which introduces all system necessary functions for the calculation (and the rendering) as virtual functions. The calculation part of the visualization method is implemented using multiple threads to support the intended parallel processing of the system. Every time the calculation part is finished an event is generated and sent to the kernel.

Kernel

The central scheduler is responsible for processing all necessary reactions of incoming user and / or calcula-

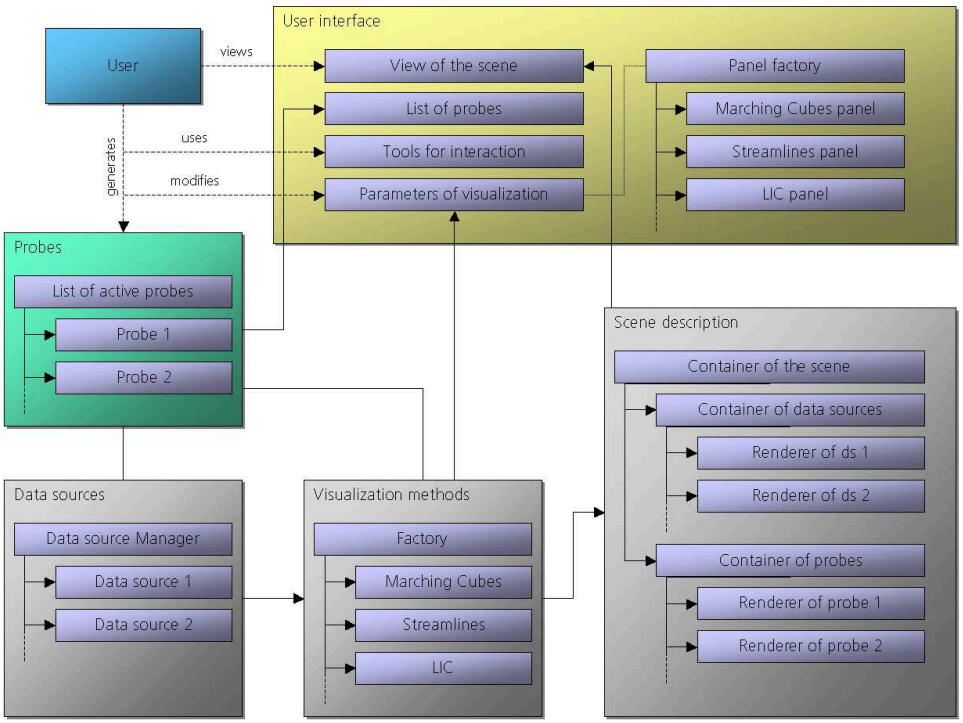

Figure 7: The basic system layout.

tion events. Every time a parameter change is generated, by user interaction or by a timer function modifying it for example, this scheduler receives a corresponding event in his incoming event queue. Being a normal thread it is then activated by the operating system having a look at his *"'incoming queue"'*. Afterwards the scheduler is responsible for initiating and controlling the necessary calculations triggered by the corresponding event. At the end the calculation-threads inform the central scheduler, again using inter-thread event communication, that the calculations have been finished. The scheduler then initiates a scenegraph node generation of the corresponding visualization method and controls the exchange of the old node(s) in the current scene description with the new one(s).

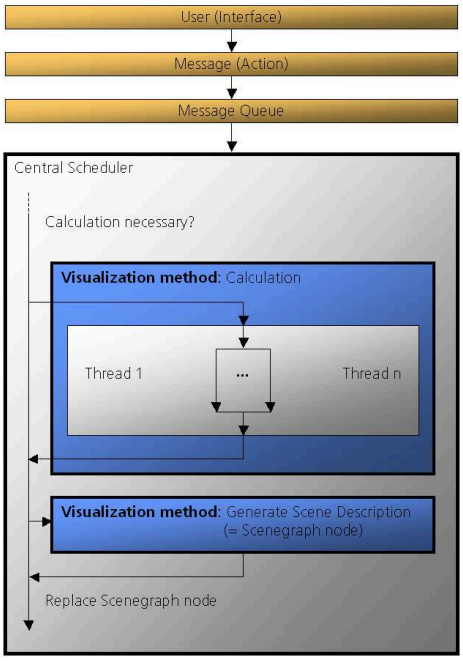

Figure 8: The central scheduler.

7. DATA REPRESENTATION AND STORAGE

In order to adapt the data source management to the resources of the system used, we developed data structures which allow us to trade off accuracy, quality and rendering performance: the *progressive grids*. The progressive grids are a special kind of *hierarchically structured grids* [Wil92], which originate from the field of computer graphics. There they are used to spatially arrange large amounts of geometry data. Progressive grids make use of the technology

of progressive data formats which are closely related to digital image processing (e.g. *Wavelet-, JPEG-Compression*). In these format the data is ordered by its level of detail. So the data associated with a given arbitrary resolution can be extracted efficiently. As a result the amount of data that has to be transmitted and/or processed can be freely adapted to variable system resources or user requirements. Geometry data, for example, can be transmitted and displayed simultanously in incremental granularity levels thus the viewer gets an impression of the geometry already with beginning of the transmission [Hop96]. While using progressive data formats in the context of CFD simulation data, we take advantage of these properties. It makes sense to arrange the data with respect to the information it contains. According to this alignment we built a hierarchy consisting of a spatial partitioning scheme [Sam84] that has so far been used in computer graphics or geometry.

With our work we introduce the principle of progressive data processing to CFD-data. Actually no grid used in numerical simulation is able to handle its data progressively. So these grids have to be converted into the progressive format to make use of them in our visualization system. For this conversion we are free to choose which cell types, partitioning schemes or error estimation schemes are used in concrete. All these parameters can be selected independently from each other and this offers a rich repertoire of possibilities. According to this, the converter is divided in two parts: A fixed one and a plug-in, which manages cell type information, its topology, interpolation schemes etc. The plug-in part can be replaced in order to implement different progressive grids (i.e. grids which use different cell types, partitioning schemes etc.).

The conversion itself works in the following way: The bounding box of the original simulation grid becomes the *root cell* of our progressive grid. Using the predefined decomposition scheme, a hierarchy of grid cells is then built up through spatial partitioning. An approximation error is computed through comparision of values interpolated within the current cell and the original grid. (This approximation is independent from the topology of the original grid.) A cell will be further partitioned, if its approximation error is the worst compared to all other currently unpartitioned cells (see fig. 9). The new cells generated in this way, represent a *"'better"'* approximation of the original grid. The whole partitioning process stops if a certain error tolerance has been reached.

The progressive grid generated in this way has a number of advantages. The maximum error within the leaf-cells in the hierarchy decreases fast. This minimizes the number of cells to be loaded at a given error

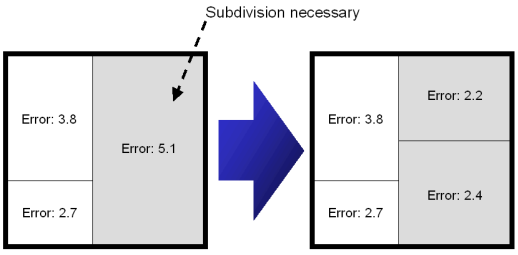

Figure 9: Subdivision: A cell will be partitioned if the error is worse compared to the error of the other cells.

tolerance. Furthermore these cells constitute a single block, because they are written in the same order their parent cells have been partitioned. The hierarchical structure of the grid can be exploited for compression in areas where low-frequency portions of the scalar fields predominate. We converted a number of datasets of fire simulations and are able to zoom through the levels of detail in real time, without making concession to performance. A visualization of streamlines (involving 200.000 vertices) using the progressive grid was comparable in speed to the one on the original, equidistant grid.

8. SUMMARY

We presented a new concept of a visualization system which is able to make use of the capabilities of modern graphic cards. This system is scalable and can be easily adjusted to different hardware conditions. Furthermore it is portable and can be easily extended with new visualization methods. Together with its capability to display scientific visualization methods together with realistic rendered it is very attractive to the end user, because he is able to investigate his simualtion results within a realistic virtual environment. The parallel approach of our system makes it very attractive for processing very large amounts of (simulation) data. Based on the progressive approach it becomes possible to visualise even large datasets on machines which have only little performance only at the cost of losing details in the loaded and displayed data.

9. REFERENCES

[Tro] Trolltech AS. Qt, c++ toolkit for application development. http://www.trolltech.com/products/qt/.

[Moo96] Robert Moorhead Aaron Trott and John McGinley. Wavelets applied to lossless compression and progressive transmission of floating point data in 3-d curvilinear grids. In *Proceedings IEEE Visualization '96*, pages 385–388. IEEE, 1996.

[OMP] OpenMP Architecture Review Board. Openmp - simple, portable, scalable smp programming. http://www.openmp.org/.

[CFX] CFX. Cfx, cfd software package. http://www.software.aeat.com/cfx/.

[Coi] Coin3D. Coin, scenegraph based 3d graphics library. http://www.coin3d.org/.

[Vos02] Gerrit VoßDirk Reiners and Johannes Behr. Opensg - basic concepts. http://www.opensg.org/OpenSGPLUS/symposium/Papers2002/.

[Fem] Femlab. Femlab, pde solver package. http://www.femlab.com/femlab/.

[Fre99] Lori A. Freitag and Raymond M. Loy. Adaptive, multiresolution visualization of large data sets using a distributed memory octree. In *Proceedings of SC99: High Performance Networking and Computing*, 1999.

[Hop96] H. Hoppe. Progressive meshes. In *SIGGRAPH '96: Proceedings*, 1996.

[ATI] ATI Techonogies Inc. Ati radeon, graphic board. http://www.ati.com/.

[Flu] Fluent Inc. Fluent, cfd software package. http://www.fluent.com/software/fluent/.

[Flo] Flomerics Ltd. Flovent, cfd based software. http://www.flovent.com/.

[MPI] mpi forum.org. Mpi - message passing interface. http://www-unix.mcs.anl.gov/mpi/indexold.html.

[Bar99] W. Bartelheimer M. Schulz, F. Reck and T. Ertl. Interactive visualization of fluid dynamics simulations in locally refined cartesian grids. In *Proceedings IEEE Visualization '99*, pages 413–553. IEEE, 1999.

[NVI] NVIDIA. Geforce, graphic processing unit. http://www.nvidia.com/.

[Rus00] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Siggraph 2000: Computer Graphics Proceedings*, 2000.

[Sam84] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2), June 1984.

[SGIa] SGI. Open inventor, object oriented 3d graphics api. http://www.sgi.com/software/inventor/.

[SGIb] SGI. Opengl, 3d rendering api. http://www.opengl.org/.

[Sch02] Sascha Schneider Thorsten May and Volker Luckas. Parallel real time fluid simulation and animation with fractal optical refinements. In *ESM 02: Proceedings of the 16th European Simulation Multiconference, Modelling and Simulation 2002*, pages 224–228, 2002.

[Mar96] Kenneth M. Martin William J. Schroeder and William E. Lorensen. The design and implementation of an object-oriented toolkit for 3d graphics and visualization. *IEEE Visualization '96*, http://public.kitware.com/VTK/:93–100, 1996.

[Wil92] J. Wilhelms and A. van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics (TOG)*, 11(3), 1992.